Introduction to Modelling

3. Programming MATLAB Functions

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.
https://github.com/JimDuggan/MATLAB

Overview

- MATLAB allows you to create your own function M-files.
- A function differs from a script file in that the function M-file communicates with the MATLAB workspace only through specially designated input and output arguments
- Functions are indispensable tools when it comes to breaking a problem down into manageable pieces

MATLAB Function y = f(x)

- Write a function that takes in a vector and returns the mean and the standard deviation.
- Make use of the MATLAB functions mean() and std()

```
function [avg, stdev] = stats(x)
  avg   = mean(x);
  stdev = std(x);
end
```

```
v = 1:5

v = 1    2    3    4    5

>> [a,s] = stats(v)

a = 3

s = 1.5811
```

General form of a function

- A function M-file *name*.m has the following general form

```
function[outarg1, outarg2, …] = name(inarg1,...)
% comments to be displayed with help
...
outarg1 = ...;
outarg2 = ...;
...
```

Challenge 3.1

- Write a function that takes in a vector and returns all the even numbers in a new vector

Challenge 3.2

- Write a function that create an adjacency matrix of random 1s or 0s, and ensures that the diagonal is zero

Further information (1/5)

- **Function keyword**
  - The function file must start with the keyword function
- **Input and output arguments**
  - The input and output arguments define the function’s means of communication with the workspace
- **Multiple output arguments**
  - If there is more than one output argument, the output arguments must be separated by commas and enclosed in square brackets
- Function names should follow the MATLAB rules for variable names
- **Help text**
  - When you type help *function_name,* MATLAB displays the comment lines that appear between the function definition line and the first non-comment line.

Further information (2/5)

- **Local variables: scope**
  - Any variables defined inside a function are accessible outside the function. These local variables exist only inside the function, which has its own workspace separate from the base workspace.
- **Global variables**
  - Variables which are defined in the base workspace are not normally accessible inside functions. If functions, and possibly the base workspace, declare variables as global, they all share single copies of those variables.

Global example

```
clear;

global X

X = 0;

inc_X();
inc_X();
inc_X();
inc_X();
inc_X();

disp('The value of X is...');
disp(X);
```

```
function inc_X()
global X
X = X + 1;
```

```
>> TestGlobal
The value of X is...
    5
```

Further information (3/5)

- **Persistent variables**
  - A variable in a function may be declared persistent.
  - These (unlike local variables) remain persistent between function calls
  - A persistent variable is initially an empty array
- **Vector arguments**
  - Input and output variables can be vectors

Persistent example

```
function persist
persistent counter

if isempty(counter)
    counter = 1;
else
    counter = counter + 1;
end

disp(['The current counter = ' ...
       num2str(counter)]);
```

```
>> persist()
The current counter = 1

>> persist()
The current counter = 2

>> persist()
The current counter = 3

>> persist()
The current counter = 4

>> persist()
The current counter = 5
```

Further information (4/5)

- **How function arguments are passed**
  - If a function changes the value of any of its input arguments, the change is NOT reflected in the actual argument
  - An input argument is only passed by value if a function modifies it.
  - If a function does not modify an input argument, it is passed by reference.
- **Checking the number of function arguments**
  - A function may be called with all, some or none of its arguments. The same applies to output arguments
  - nargin displays the number of arguments passed
- A variable number of arguments can be passed.

Further information (5/5)

- Subfunctions
  - The function M-file may contain the code for more that one function. The first function is the primary function.
  - Additional functions are known as subfunctions, and are visible only to the primary function and the other subfunctions.
  - Subfunctions follow each other in any order AFTER the primary function.
- Private functions
  - A private function is a function residing in a subdirectory with the name **private**. They are only visible to functions in the parent directory.

Challenge 3.3

- Write a function that takes a matrix and converts it into a row vector

Challenge 3.4

- Write a function that calculates the scalar product of two vectors.

Function Handles

- A handle can be created to a function using @

```
fhandle = @sqrt
```

- Then the function feval can be used to activate the function through its handle

```
feval(fhandle, 2) or
fhandle(2)
```

- This a very useful: we can now effectively pass functions to functions

Code Example

```
function [answ] = misc(fH, data)
answ = feval(fH, data)
```

```
>> misc(@sqrt,100);

answ =

   10

>> misc(@sin,100);

answ =

  -0.5064
```

Anonymous Functions

- Provide a quick means of creating simple functions without having to create M-files each time
- Can be constructed at the command line or in a script file
- Syntax:

```
fhandle = @(arglist) expr
```

fhandle = @(arglist) expr

- **expr** represents the body of the function, and consists of any single, valid MATLAB expression
- **arglist** is a comma-separated list of input arguments to be passed to the function.
- **@** constructs a function handle, and is required as part of an anonymous function definition.
- The LHS (function handle) can then be used just like any MATLAB

Example (1 parameter)

```
f = @(x) x.*2

f(1:10)
```

```
f =

  function_handle with value:

    @(x)x.*2


ans =

     2     4     6     8    10    12    14    16    18    20
```

Example (two parameters)

```
g = @(x,y) sum(x.*y)

g([1 2 3], [4 5 6])
```

```
g =

  function_handle with value:

    @(x,y)sum(x.*y)


ans =

    32
```

Example – returning a matrix

```
h = @(x) [x .* 2; x.^2]

h(1:5)
```

```
h =

 function_handle with value:

   @(x)[x.*2;x.^2]


ans =

    2    4    6    8   10
    1    4    9   16   25
```

Challenge 3.5

- Write an anonymous function that returns the response to the equation of a line
- Inputs should be:
  - An x vectors of line points
  - The slope of the line (m)
  - The intercept (c)
- Plot the results on a graph

Challenge 3.6

- Write an anonymous function that takes in a vector and returns all the odd numbers in a new vector